Práctica 2. Programación dinámica

Francisco Javier Molina Rojas
javier.molinarojas@alum.uca.es
Teléfono: 722528757
NIF: 45386606Q

16 de diciembre de 2022

1. Formalice a continuación y describa la función que asigna un determinado valor a cada uno de los tipos de defensas.

$$f(salud, ataqueporseg, danio, dispersion, coste) = \frac{salud + ataqueporseg + danio + dispersion}{coste}$$

Para realizar la funcion que dara un valor a cada defensa, deberemos tener en cuenta los atributos de esta, intentando sacar los maximos de estos en relación a su coste. Los atributos que dan valor a la defensa son: los ataques por segundo (attacksPerSecond), el daño (damage), la dispersion (dispersion) y la vida (health). El atributo que resta valor a la defensa es su coste (cost). Entonces haremos un promedio devolviendo la suma de los atributos que dan valor a la defensa entre su coste.

Funcion DefenseValue

```
float DefenseValue(std::list<Defense*>::iterator currentDef)
{
        return ((*currentDef)->health + (*currentDef)->attacksPerSecond +
        (*currentDef)->damage + (*currentDef)->dispersion)/(*currentDef)->cost;
}
```

2. Describa la estructura o estructuras necesarias para representar la tabla de subproblemas resueltos.

Para representar la tabla de subproblemas resueltos, vamos a usar una matriz de flotantes de dimension numdecandidatos(defensas) * presupuesto + 1 (ases+1). Esto lo hacemos con el objetivo de tener una matriz que vaya guardando el maximo valor posible almacenable en la mochila.

Estructura de datos seleccionada para la tabla de subproblemas resueltos

```
        float evaluacionTotal[Candidatos.size()][ases+1];
```

3. En base a los dos ejercicios anteriores, diseñe un algoritmo que determine el máximo beneficio posible a obtener dada una combinación de defensas y *ases* disponibles. Muestre a continuación el código relevante.

Algoritmo para rellenar la tabla de subproblemas resueltos

```
void DEF_LIB_EXPORTED selectDefenses(std::list<Defense*> defenses, unsigned int ases,
                std::list<int> &selectedIDs, float mapWidth, float mapHeight, std::list<Object*> obstacles)
{
    //definicion de variables necesarias
    //creamos vector de candidatos
    std::vector<candidato> Candidatos;
    //iterador para recorrer las defensas
    std::list<Defense*>::iterator defit = defenses.begin();

    //No contamos el centro de extraccion ya
    //que este es necesario para el funcionamiento
    //metemos la primera defensa(centro de extraccion)
    selectedIDs.push_back((*defit)->id);
    //pasamos a la siguiente defensa
    defit++;


    //bucle para inicializar la lista de candidatos
    while(defit != defenses.end())
    {
```

```
        //Cada candidato tendra un puntero a la defensa y valor que tiene la misma defensa
        Candidatos.push_back(candidato((*defit),DefenseValue(defit)));
        defit++;//pasamos a la siguiente defensa
    }
    //ordenamos el vector (de menor a mayor coste)
    std::sort(Candidatos.begin(),Candidatos.end());


    //utilizaremos una matriz de flotantes para poder representar la tabla de subproblemas resueltos
    //ademas, esta debera ser de dimension
    //numdecandidatos(defensas) * presupuesto + 1 (con el objetivo de poder llegar al valor de ases
    float evaluacionTotal[Candidatos.size()][ases+1]; //definicion de tabla de subproblemas

        //Bucle para recorrer la tabla de subproblemas
    for(int i = 0 ; i < Candidatos.size() ; i++)
    {
        for(int j = 0 ; j < ases+1 ; j++)
        {
                //si estamos en la primera fila y j (presupuesto actual) es mayor o igual a
                //lo que me cuesta la defensa, entonces la meto en la "mochila"
            if(i == 0 && j >= Candidatos[i].def->cost)
                evaluacionTotal[i][j] = Candidatos[i].valor;

            //si estamos en la primera fila y j (presupuesto actual) es menor a
            //lo que me cuesta la defensa, entonces no meto nada en la mochila
            if(i == 0 && j < Candidatos[i].def->cost)
                evaluacionTotal[i][j] = 0;

            //si estamos en una fila superior a la primera
            if(i > 0)
            {
                //si j (presupuesto actual) es menor a el coste de la defensa actual
                //entonces no puedo meter la defensa actual (me quedo igual que en la fila anterior)
                if(j < Candidatos[i].def->cost)
                    evaluacionTotal[i][j] = evaluacionTotal[i-1][j];

                //si j (presupuesto actual) es mayor o igual a el coste de la defensa actual
                //entonces nos quedamos con el maximo de lo que tenia antes(fila anterior) y
                //lo que tendria su meto en la mochila si meto la defensa actual en la mochila
                else
                    evaluacionTotal[i][j] = std::max(evaluacionTotal[i-1][j],
                    evaluacionTotal[i-1][j-Candidatos[i].def->cost] + Candidatos[i].valor);
            }
        }
    }
    //el maximo valor disponible para el conjunto de defensas estaria en la ultima posicion
    //tanto de fila y columna de la tabla de subproblemas resueltos
```

4. Diseñe un algoritmo que recupere la combinación óptima de defensas a partir del contenido de la tabla de subproblemas resueltos. Muestre a continuación el código relevante.

Algoritmo para rellenar la tabla de subproblemas resueltos

```
void DEF_LIB_EXPORTED selectDefenses(std::list<Defense*> defenses, unsigned int ases, std::list<int> &selectedIDs
        , float mapWidth, float mapHeight, std::list<Object*> obstacles)
{
    //definicion de variables necesarias
    //creamos vector de candidatos
    std::vector<candidato> Candidatos;
    //iterador para recorrer las defensas
    std::list<Defense*>::iterator defit = defenses.begin();

    //No contamos el centro de extraccion ya
    //que este es necesario para el funcionamiento
    //metemos la primera defensa(centro de extraccion)
    selectedIDs.push_back((*defit)->id);
    //pasamos a la siguiente defensa
    defit++;


    //bucle para inicializar la lista de candidatos
    while(defit != defenses.end())
    {
        //Cada candidato tendra un puntero a la defensa y valor que tiene la misma defensa
        Candidatos.push_back(candidato((*defit),DefenseValue(defit)));
        defit++;//pasamos a la siguiente defensa
    }
    //ordenamos el vector (de menor a mayor coste)
    std::sort(Candidatos.begin(),Candidatos.end());


    //utilizaremos una matriz de flotantes para poder representar la tabla de subproblemas resueltos
    //ademas, esta debera ser de dimension
    //numdecandidatos(defensas) * presupuesto + 1 (con el objetivo de poder llegar al valor de ases
    float evaluacionTotal[Candidatos.size()][ases+1]; //definicion de tabla de subproblemas

        //Bucle para recorrer la tabla de subproblemas
    for(int i = 0 ; i < Candidatos.size() ; i++)
    {
        for(int j = 0 ; j < ases+1 ; j++)
        {
                //si estamos en la primera fila y j (presupuesto actual) es mayor o igual a
                //lo que me cuesta la defensa, entonces la meto en la "mochila"
            if(i == 0 && j >= Candidatos[i].def->cost)
                evaluacionTotal[i][j] = Candidatos[i].valor;

            //si estamos en la primera fila y j (presupuesto actual) es menor a
            //lo que me cuesta la defensa, entonces no meto nada en la mochila
            if(i == 0 && j < Candidatos[i].def->cost)
                evaluacionTotal[i][j] = 0;

            //si estamos en una fila superior a la primera
            if(i > 0)
            {
                //si j (presupuesto actual) es menor a el coste de la defensa actual
                //entonces no puedo meter la defensa actual (me quedo igual que en la fila anterior)
```

```
                if(j < Candidatos[i].def->cost)
                    evaluacionTotal[i][j] = evaluacionTotal[i-1][j];

                //si j (presupuesto actual) es mayor o igual a el coste de la defensa actual
                //entonces nos quedamos con el maximo de lo que tenia antes(fila anterior) y
                //lo que tendria su meto en la mochila si meto la defensa actual en la mochila
                else
                    evaluacionTotal[i][j] = std::max(evaluacionTotal[i-1][j],
                    evaluacionTotal[i-1][j-Candidatos[i].def->cost] + Candidatos[i].valor);
            }
        }
    }
    //el maximo valor disponible para el conjunto de defensas estaria en la ultima posicion
    //tanto de fila y columna de la tabla de subproblemas resueltos

        //variables para recorrer hacia atras los resultados de la mochila
    int j = ases, i = Candidatos.size() - 1;
    //lista de defensas seleccionadas (defensas que esten en la mochila)
    std::list<Defense*> DefensasSeleccionadas;
    //mientras j (el presupuesto) sea mayor que 0 y la fila sea mayor igual a 1
    while(j > 0 && i >= 1)
    {
        //si la posicion actual es distinta a la superior (se ha metido la defensa)
        if(evaluacionTotal[i][j] != evaluacionTotal[i-1][j])
        {
                //guardamos la defensa
            DefensasSeleccionadas.push_back(Candidatos[i].def);
            //al presupuesto se le quita lo que nos costo la defensa
            j = j - Candidatos[i].def->cost;
        }
        i--; //pasamos a la fila superior
    }

        //Ahora insertaremos las IDs de las defensas seleccionadas
    std::list<Defense*>::iterator dit = DefensasSeleccionadas.begin();
    while(dit != DefensasSeleccionadas.end())
    {
        selectedIDs.push_back((*dit)->id);
        dit++;
    }
}
```

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de este documento confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.